

SISTEM PROGRAM LAMA

Yrd. Doc. Dr. Huseyin Pehlivan

username: merue

password: 911abc

Derom konusu:

A) Unix Sistemlerde Script Yazılım

B) Sistem G  r  l  r  yle Problem Yazılım

① Unix

Bir işletim sistemidir. Kullanıcı ile bilgisayar arasındaki kurallar zamarıdır. Bilgisayar kaynaklarını yönetir (files, programs, disks, network)

Modern işletim sistemidir. Kararlı, yapılandırılabilir, faklı kullanıma a  k.

Bir  ok bilimsel ve end  striyel alanda kullanılır, kaynak kodu a  ktır, programlama   uresi gelişmiştir.   cretsiz sahip alınabilir.

D  nyada %65 Unix / Lmux kullanılıyor.

Unix 1970'lerde Bell Lablarda Ken Thompson ve Dennis Ritchie tarafından programcılar tam geliştirilmiştir.

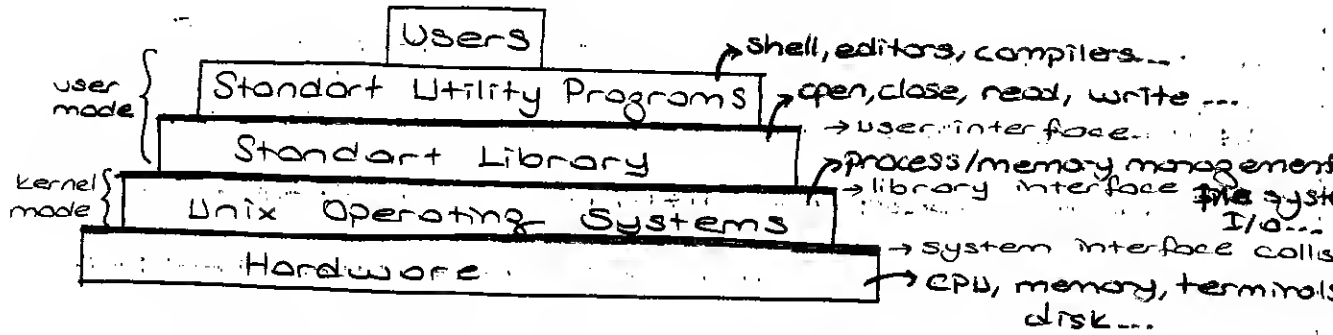
Ken B'de program yazıyor, B yetersiz kalınca Dennis B'den C dilini   retip UNIX'i bununla yazıyor. %95'i C, %5'i (kernel) assembly ile yazılmıştır.

BSN → Unix sisteminin 2 ayrı koldan gelişti  i, ba  langı  ta tek bir sistem oldu  ğunu g  steriyor.   r  : SunOS 4, Ultrix

SYS4 →

de

Unix Tabanlı Sistemde Tabakaları:



İfopen dersinde library interface'i,

open dersinde sistem call interface'ini kullanmış oluruz. İki emri de aynı kod içerisinde kullanabiliriz.

Kernel: Sistem donanımını kontrol eden ve çeşitli alçak seviyeli işlemleri kontrol eden bir yazılımdır.

İstek kernela bildirilir, kernel servisleri tahsis eder.

Shell: Kullanıcı seviyeli programdır. Kullanıcıdan aldığı komutları bildirir, komutların icrası sağlar ve sonlandırır, asıl görevi budur. Komutu kernel kollar.

UNIX'te 200'den fazla utility programları ya da komutları bulunur. Bunlar OS'nin parçası değildir, yine de kullanıcının kullanımına uygun komut-

lardır (dosya işlemleri, hesaplamalar ve yazılım geliştirme araçları gibi). Uzakta UNIX'e bağlanıldığında bize bir terminal sunulur, UNIX sistemine login yapmak (bağlanmak) bir accounts (hesaba) ihtiyacı duyar. Hesap kullanıcısı ismi ve şifresi ile ilişkilendirilmiştir. Login: mit ile kurulan mit OS çalıştırıldığı ilk olarak oluşturulan süreçtir. PID (process ID)'si 1'dir. mit bir login işlemiyle karşılaştığında bir child süreç oluşturulur. getty programı user name ister. Alındığında login yapıldı. Login passwordun geçerliliğini kontrol eder. Geçerliyse login shell oluşturulur.

Kullanıcı tam belirlenmiş shell kullanıcısının kullanımına açılır. Login metodu x display'i de kullanılabilir. xdm'e alternatif olarak gdm ve kdm verilebilir. Ama xdm her türlü sistem yöneticisi tarafından kurulmalıdır. Default değerlerdir. xdm otomatik olarak kurulumda gelen masquitos programlarını destekleyen bir şeydir.

Güvenli Erişim Araçları

Terminal bağlantısı için PuTTY (on Windows)

MindTerm (Java applet)

Masaüstü erişimi için X-Win32 → CDE, KDE, GNOME

Weirdx (Java applet)

Dağıtım erişimi için

?

Shell Prompt

\$ / % / # / username@hostname > / hostname%

Komutları kullanılabilmek

Shell

Kullanıcıların işletim sisteminde program çalışabilmesini sağlayan programdır

who → şu an çalışan programcılar

date → Sistemin tarihini ve saatini gösterir

ls → şu an bulunan dosya ve klasörler

cat textfile → ?

ls -l → klasördeki öğeler hakkında daha fazla bilgi

Command Options Argument(s)

Komutun davranışı aynıdır. Optionlar gösterilen bilgiyi komutun davranışının kapsamı alanını değiştirir. Örneğin birden çok dosyayı 1 komutla silmek istiyorsak option kullanmalıyız.

Ör1 ls tgm ls-l

ls-o

ls-la

ls-o; ls-l

ls-f

ls al textfile 1

ls al textfile 1 textfile 2

Eğer shell prompt verilmece demek ki son verilen program hala devam ediyor, illa prompt istiyorsak CTRL-C ile programın terasına sonlandırılır, CTRL-Z ile programın terasına ara verilir. Daha sonra kaldığı yerden devam ettirebiliriz, quit, bye, exit gibi komutlarla da bunu başarabiliriz.

Logging Out

exit ile sistemden çıkılabilir, kullanıcılar terminale bağlı olarak logout veya CTRL-d'de kullanabilir.

② UNIX'te Dosya Sistemi

Fiziksel depolama alanına bir interface'dir. Aynı zamanda I/O cihazlarla da bir orayışa oluşturur.

UNIX'te herşey bir dosyadır. UNIX'teki herşey bir dosyaya erişir gibi erişebiliriz. (Soket, terminal, başka bir makinedeki açık bir dosya, vs. olabilir)

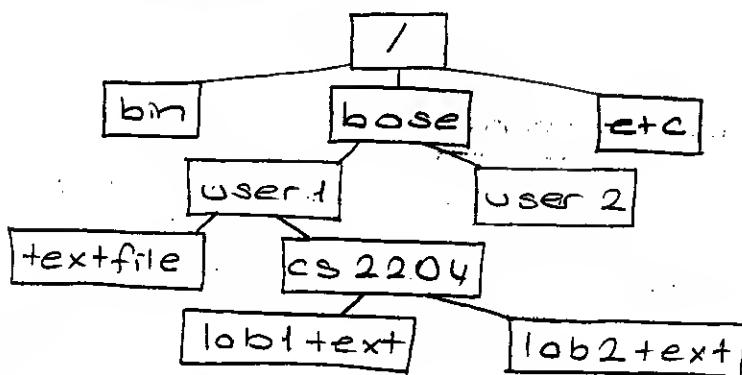
Directory: Diğer dosyaları içeren basit bir dosya. UNIX sisteminde, sürüş harfleri kullanılmaz.
Working directory: Dosya sistemi üzerinde bulunulan o anki konum.

pwd (print working directory): İlk directoryden working directory kodu gelinen yolu (absolute path - tam yola dizini) yazdırır.

Konulan her bir komut bulunulan dizini alır. (Eğer başka dizinin ismini parametre olarak vermediysek).

home directory: Sisteme login olduğun bize sunulan dizin. Home dizin ismi .n karakteri kullanılır. (n user id).

root directory (/):



absolute path → roottan bulunulan yere kadar

Ör. /home / user / textfile

~ user / textfile

~/textfile

relative path: Bulunulan dizinden başlanarak erişilebilecek yol. • bulunulan dizini; • • bir üst dizini gösterir.

Ör. textfile

CS2204/ lab1.txt

Bazı Standart Dizimler

/ Root directory

/bin Dizin altındaki standart komutlar (executables)

/dev Dizin içerisindeki blok ve karakter tabanlı sürücülere linkler

/etc Host'a özel yapılandırma ve servisler

/home Kullanıcıların home dizinleri; günümüzde export dizini altına yerleştiriliyor.

/lib Geçitli programlama dilleri için library dosyaları

/sbin Sistem komutları ve utilitiler (boot için gereken)

/tmp Temporary dosyaları (bazı programlar tam geçici dosya üretir. Örneğin edit yaparken)

/usr User utilitileri ve uygulamaları /usr/local

/var Boyutu değişebilen dosyaların yerleştiği dizin (system log, spools, emails, yazıcı dosyaları)

changing Directories

cd - Bulunulan dizini değiştirmeye yarar.

Ör cd / home / user1

cd /... /... / user1

ls -lF'nin Çıktısı

↓	↓					
file type	number of hard links					
permission	users	group	file size	modified date	file name	

plain (-) → Çoğu dosya, binary veya text.

directory (d) → diğer dosya kümelerini gösteren dosya

link (l) → diğer dosya ve dizinlere pointer

special (b) → blok cihaz (diskler, CD-ROM)

(c) → karakter cihaz (klavye, joystick)

• touch <file> ile yeni dosyalar ya da modify yapılır.

• mv <file1> <file2> ①

mv <file1> <dir> ②

mv <file1> <dir/file2> ③

mv (move) ile dosyayı hareket ettirir ve ismini değiştirir.

- | | |
|-----------------------|---------------|
| ① dosya adını | } değiştirir. |
| ② dosya yeri | |
| ③ dosya adını ve yeri | |

• cp <file1> [<file2> | <dir> | <dir/file2>].

cp (copy) kopyalama yapar.

• rm [-i] <file(s)>

rm (remove) dosya(ları) siler. [-i] varsa onay ister.

• mkdir <directory-name>

mkdir (make directory) dizin oluşturur.

• rmdir <directory-name>:

rmdir (remove directory) dizin boşsa siler, değilse boşaltır.

• rm -r <directory-name>

-r olursa dizini içindekilerle beraber siler.
recursively

• ln -s <existing-file> <link-name>

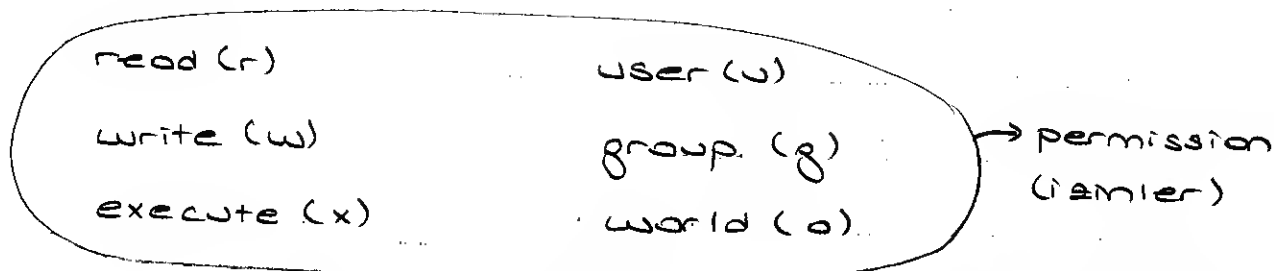
ln (link) link oluşturur. -s yoksa hardlink, varsa

softlink oluşturur. -s ile içerisinde yol bulur.
symbolic link

non. yeni bir dosya oluşturulur.

File Ownership

Chown



change mode
`chmod <mode> <file(s)>`

`chmod 700 textfile`

rwX
1 1 1 = 7
user

rwX
0 0 0 = 0
group

rwX
0 0 0 = 0
world

`chmod g+rw textfile`

ugo +/- rwX

group kısmına rw eklenir.
1 1 0 = 6

$700 + 60 = 760$ octals

- File Modification Date

Değişik dizinlerde dosyanın birçok kopyası varsa ya da dosya birkaç kullanıcı tarafından kullanılıyorsa last modification date gerekir.

touch → modification date'i update eder.

cat textfile1 textfile2 → dosya içeriklerini (bir defada) gösterir.
concatenate

Her dosya tek bir owner'a sahiptir. chown ownerı değiştirmede kullanılır. Genellikle sadece root bu komutu kullanır.

less / more + textfile → dosya içeriklerini gösterir (sayfa sayfa page-down ile)

Wild Cards in File Names

- * herhangi bir dosya
- ? herhangi bir karakter
- [] karakter aralığı vermek için ör: [a-c]=a,b,c

Other File Systems

- disk-based (harddisk, CDROM, floppy disk)
- distributed (networke dağıtılmış kaynaklar)
- pseudo (memory tabanlı)

③ Text Editing

cp (copy), mv (move), rm (remove), ln (link), cat, less.

Why vi?

Komutlar klavye ile girilir, Text tabanlı bir editör olduğundan mouse kullanılmamalıdır. (ASCII text)

Vi (improved) = Vim

vi Basics

vi [filename(s)]

Windowu kontrol eder. Mode tabanlı bir editördür.

2 moda vardır:

- Command mod

- Insert mod

Genelde komutlar tek karakterlidir ve büyük KÜ-
çük harf ayrımı vardır.

Modlar arasında geçiş. Tüm geçişli karakterler yazılır.

command → Insert. a ya da o

insert → command ESC

Command Mod

VI editörü her zaman komut moduyla çalışır.

Commandlar ① cursor'ü hareket ettirme

② değiştirme

③ dosya operasyonları

④ arama

⑤ insert moda girip

şeklinde 5 sınıfa ayrılır.

① Cursor'ü hareket ettirme:

← ↑ ↓ →
h j k l

n goklu ilerlemede kullanılır.

n (j | k | l | h)
↓
sayı

② u son değişikliği geri alır

x karakteri siler.

dd satırı siler

dw kelimayı siler

rx karakterle x'li yer değiştirir.

yy satırı , kopyalar.

P kayyolanmış/ siltlanmış parçaları yapıstırın.

İ iki satırı birleştirir.

③ 22 yasa ve s. kor.

iwg yozar ve qikar.

1144 1920

File name dosya adını yazar.

19 daaemelden gik.

191 kaydetmeden çık.

!e <filename> başka bir dosya, dizelt.

In Br sonraki dosyayı dzeelt.

④ /string yazılan stringi or.

?string yazılan stringi genye döfərür.

n önceki bromoalkenler yasınaklaşma ve deben ettiler

N önceki oramanın tersine orama devam ettirilir.

% $\{, [, \}$ ile işaretlerini 'arar.

⑤ i kursorün önünden inserte başlanır

1 satır başından inserte bağlanır

9 kursår den som oppende begjær

A satırından itibaren başlanıyor...

0 kursorun altında yeni satır oluşturur.

0 kursörün üstünde yeni satır oluşturur.

cu kelimeyi yer değiştirin

C satır sonunu değiştirir.

Chapter 8'de diğer komutlar listelenmiştir.

v haricindeki büyük karakterleri yazarken dikkat etmeliyiz.

CTRL-X ile editörden çıkılır.

İmax Unix sistemlerindeki en yüksek editörlerden biridir. Yazılım geliştirmede tercih edilir. Kullanılan dilin syntaxına göre keywordleri tanıyacak şekilde. Grafik arayüzü bulunur. Bu editörde CTRL-X sonra CTRL-C ile çıkılır.

~\$ vim ~\$ gvim su

~\$ echo \$SHELL

~\$ echo \$DISPLAY

④ 4. Dem. Redirection

Three Standard Files

Her bir terminal için standart dosya tanımlanmıştır.

stdin: Giris karakter stream	} terminalde
stdout: Çikis karakter stream	
stderr: Error mesajı olan yer	

Redirection stdout

Yönlendirmek için > karakteri kullanılır.

>> ise mevcut bir dosyaya sonuna itibaren yazma yapar.

\$ date > 222

Bu date komutunun verdiği çıktıyı 222'ye yazar.
(Böylelikle, 222'nin içeriğini silerek).

more 222

ile 222'ye bakarsak bunu görürüz. Bu komut
londan önce 222 vardıysa üzerine yazar, yoktuysa
yeni oluşturur.

\$ PWD → çalışma dizinin km yolları için
cat file1 >> file2

Redirect std in

Giriş < karakteri ile yapılır. Bu karakter sayesinde
baska denetclerden giriş alabiliriz.

mail user@domain.com < message.txt

at 3am < cmds or at 0945 < cmds

sort < friends > sorted_friends

interactive_program < command_list

Pipes (|)

Bir komutun stdoutunu diğerinin stdin'ine bağlamak.

ls -la | less

ls -al | wc

cat file | wc

Processes (|)

Aynı anda birden fazla program çalışır.

date; who. ls -al & wc *

Önce date sonra who komutu. Tercih edilir,

& karakterini önce alan komut kendinden sonra gelen komutla eş zamanlı olarak çalışır.

Filters

wc kelimelerin/satırların/karakterlerin sayısını verir.

grep verilen pattern (RE) uygun satırları arar.

grep -<pattern> <filename>

sort Sıralama yapar.

-r tersten sıralar

-n numerik sıraya göre sıralar.

+2n 2. sütuna göre sıralar.

cut -c1-5 Satırın ilk 5 karakterini gösterir.

-c1,5 Satırın ilk ve 5. karakterlerini gösterir.

-d1-f1,5/etc/passwd: user ID'yi name'e map eder.

head Dosyanın ilk n satırını görüntüler.

head -n <filename>

tail -n <filename> Dosyanın son n satırını görüntüler.

+n <filename> Dosyanın ilk n satırından sonrasını görüntüler.

diff İki dosyanın farklı olan satırlarını gösterir.

cmp İki dosyanın farklı ilk satırını gösterir.

diff <file1> <file2>
cmp

od Dosyanın içeriğini 8li tabanda gösterir.

`od -c`

0000010 → end of file (EOF)'in gösterimi.

ls-lt modification zamanına göre dosyaları listeler.

crypt verilen bir key'e göre dosya içerikleri şifrelenir.

`crypt key <clear. file> encrypted file`

tr girdideki karakterleri verilen parametrelere göre başka bir karakter setine dönüştürür.

`tr "[lower]" "[upper]" < <file>`

lower caseleri upper case'ye dönüştürür.

uniq Tekrarlanan satırlarda işlem yapar.

`uniq -d <file>` tekrarlananları yazdırır.

`uniq -u <file>` tekrarlanmayacak şekilde tekrarlananların sadece 1 tanesini yazdırır.

Process İsmi Komutları

ps Aktif processleri listeler.

top Processlerin sistem kullanımını dinamik olarak gösterir.

kill Processi sonlandırır.

time Bir process için zamanlama bilgilerini sağlar.

Communication.

talk Sistemdeki, diğer kullanıcılarla mesajlaşmada kullanılır.

talk smith pts/2

write Başka bir kullanıcıya mesaj göndermede kullanılır.

write smith pts/2

mesg [n/y] : mesajı reddetmek ya da izin

vermek.

mail, pine Text tabanlı email programıdır.

ftp, sftp Text tabanlı ftp programıdır.

Bu anda sadece sftp'ye izin veriliyor.

ftp portu kapalı.

telnet, ssh Diğer makinelere direkt bağlantı için.

lynx Text tabanlı web browser

⑤ Regular Expressions

grep dosya ve içindeki metinleri aramak için.

egrep regular expressionları kullanmak için

vi/emacs text editors

ex satır editör

sed akış editör

awk pattern tanıma dili

perl yazı dili

Temel regular expressionlar içinde bu meta-karakterler özel anlamlara sahip değildirler.

?, +, {, }, (,), |

Grep ile beraber kullanılırlar.

\?, \+, \{, \}, \(\, \), \|

şeklinde kullanılırız.

Biz regular expressionları egrep ile kullanırız.
egrep pattern filename(s)

ör egrep "abc" textfile → abc'yi içeren satırları ekrana basar

egrep -i "abc" textfile → Karakterlerin büyük veya küçük olmasını dikkate almaz

egrep -v "abc" textfile → abc'yi içermeyen satırları basar.

egrep -n "abc" textfile → satırları numaralı olarak gösterir.

Meta Characters

period (.) tek bir karakteri bildirir.

ör: "a.c" → abc, adc, a&c, a;c, ...

"u.x" → unix, urax, u3cx, ...

asterisk (*) sıfır veya çok sayıda tekrarı ola-

bileceğini gösterir. (shell'deki wildcard larla karıştırma)

ör: "ab*c" → ac, abc, abbc, abbbc, ...

".*" → 0 veya çok sayıda . → herhangi bir kelime

plus (+) RE'nin bir veya daha çok sayıda olma-
cağını gösterir.

ÖR: "ab+c" → abc, abbc, abbbbc, ...

question mark (?) RE'nin 0 veya 1 kez kullanıldığını gösterir.

ÖR: "ab?c" → ac, abc, ...

logical (|) Birden önce ya da sonra bulunan RE'den aynı aynı paternler üretilmesini sağlar.

ÖR: "ab|de" → ab, de

caret (^) Satır başı anlamındadır.

ÖR: "^D.*" → Satırın D ile başlaması gerekir.
ğini söyler.

Dollar sign (\$) Satırın sonuna belirtir.

ÖR: ". *d\$" → d ile biten satırları gösterir.
".*" olmasa da aynı olurdu.

Backslash (\) Kendinden sonra gelenin bir karakter olarak gösterileceğini belirtir.

ÖR: /\.txt → .txt

square brackets ([]) Üretilen bir patern içinde bulunmasını istediğimiz karakterleri liste olarak girer.
memizi sağlar.

ÖR: b[a e]k → bak, bek.

parentheses (()) Gruplama yapmak kullanılır.

ör $a(b^+)$ = abc, abcba, abcbaabc...

braces ({}) Bir RE'nin kaç kez tekrarlanacağını belirtir.

ör $[a-z]\{3\}$ → 3 kşşık harften oluşan kelimeler.

ör - \$ egrep -n "o [^abc]" BASH

Bu örnekte a ile başlayan abc dışındaki karakterlerden biri ile devam eden kelimeler ekrana bastırılır.

* Başlık da özellikle belirtilmiş olabilir.

"2 kez" örnekten "2kez" yazdırılması.

word searching with egrep

• /usr/dict/words

Bunun içine alfabetik sırada bulunan 5 sesli harf içeren kelimeleri bulmak için RE.

$^[\text{aeiou}]^*o[\text{aeiou}]^*e[\text{aeiou}]$

İçinde bütün sesli harflerin bulunduğu alfabetik sırada olan kelimeler.

$^[\text{aeiou}] = [bcdf...]$

* egrep -f → Komutun, verilen RE expressionun dosyadan okuması gerektiğini söyler.
RE'i alphavowels
↓
bu dosya içinden olması gerektiğini söyler.

egrep -f. monoton

6. ya da daha fazla kelimelerin alfabetik sıradan da yer alır. Karakterleri alfabetik sırada uyarıya alabiliriz ? kagoriz

... a?b?c?

grep "..."

6) UNIX Window System

Window sistemleri OS üzerinde bulunan ve kullanıcıya grafiksel arayış sunan sistemlerdir. (GUI) windows (pencereler), Icons (ikonlar) ve interrupt driven interaction tabanlıdır.

Window sistemleri geliştirmeden önce Unix sistemleri geliştirilmişti. Böyle olduğundan komut satırını etkin kullanmak için UNIX kendi araçlarını geliştirmiştir. Bunlardan biri text tabanlı arayış sunan SHELL'lerdir. Windowtan önce UNIX text tabanlı kontrolör SHELL'lerle gerçekleştirir. Bu yüzden onlarca SHELL var. UNIX sistemlerinde window sistemleri, X1126 gibi. Herhangi bir UNIX sistemi ve windows sisteminin birlikte kullanılması için X1126 lib-rarisi ile window uyumlu çalışmalıdır.

X-Server

Bunu sağlamak için X-server hizmet verir. Bu hem konsol üzerinde hem de remote olarak çalışma imkanı kullanıcılara verir.

- Minimum bir interface iserir. (display...)
- Makina üzerinde de gerçekleştirilebilir.
- Display yönetmek için servisler, sahiptir.
- Basit grafik işlemleri destekler. Basit rectangular alanları bilgileri clientlara tahsis edebilir.
- Client-server mimarisme uygun bir yazım.

The X Client

X client - X server'in iki tarafı da aynı makine üzerinde bulunabilir. Baska bir makinedeki X serverden de hizmet alınabilir.

Client üzerinde yapılmaya çalışılan her bir işlem komut olarak sunucuya iletilir. Sunucu görsel bir etkiye sahiptir bu client üzerinde yansır.

Sanki orada X server yokmuş gibi makine ile client etkileşimde bulunuyormuş gibi etkiye sahiptir.

Window Manager

Window sisteminin yönetimiyle ilgilenen yazılımlar. Borderlar, sliderlar, tıgerebilir.

pencere yöneticileri:	kwm	metacity	mwm
pencereler:	↓ KDE (default)	↓ GNOME (default)	↓ motif standalone

Desktop environment

KDE, Xfce, GNOME

Bunlar default olarak OS kurulumu ile gelmezler. Bunları ayrıca kurmamız gerekir. Dosya yöneticileri (Nautilus), ikonları, panelleri, konfigürasyon araçlarını ve appletleri içerir.

\$ morec, X-micrre

⑦ UNIX Shell Environments

Shell kullanıcı ile OS arasında bir komut satırı ortaktır. Sisteme bağlanınca otomatik olarak sunulur. Komut getirici olmakla birlikte bir programlama dili olarak düşünülebilir. Bu dil yardımıyla shell scriptler yazılır. Bunlar sıradan bir dosya gibi görülebilir.

Shell Interactivity

- Komut satırı üzerinde tetiklemeye yardım eder.
- Birilen bu komutun opsiyonları ve argümanları belirtmek için parçalara ayrılır.

- Kullanıcıya Gollama otomatik sunar.
- Komut satırı üzerinde komut ve arguman-
ları girilirken argumanın otomatik olarak tanımlama özelliği verir.
- Yazımı uzun zaman alan komutlar için kısa
isimler tanımlanabilir.
- Yazılan komutları görmek için yapılandırma yapar.
- Eski komutları tutar, komut satırını düzenleme imkanı sunar.

Shell Programming

- Değişkenler
- Shell scriptler
- Kontrol yapıları (döngüler ve şartlar)
- Next lecture
ders
- Fonksiyon tanımlama ve çağırma

Various Unix Shell

sh (Unix'in orijinal shelli. İlk bu yerleştirilmiştir)

ksh (Korn shell)

csh (C shell)

trsh

bash (Bourne again shell) (Linux'un orijinal shelli)

Bourne Again Shell (bash)

Biz bu shell'den yazacağız. Diğer shell'lerden de bir
çok özelliği bize sunar. Superseti sh ile gösterilir.
GNU projesinin parçasıdır.

Environment Variables

Değişken listesini "env" komutu ile görebiliriz.
Bir değişkenin değerini "echo \$varname" komutu ile öğrenebiliriz.

Değişkene değer atamasını C dilinde syntaxına uygun olarak atarız. \$ da kullanırız.

ÖR \$HOME → Kullanıcının home değişkenini gösterir.

\$PATH → Komutların bulunduğu dizinleri tutar. Bir-
dizimite komut bu dizinlerde yoksa komutu
kalmaz.

\$PS1 → Shellin verdiği komut satırını kontrol eder.

ÖR $10@1h1 \sim \$$
↓ ↓
Kullanıcı Host
İsmi name

\$USER → Kullanıcı ismini gösterir.

\$HOST → Host ismini gösterir. Bazı noktemelerde
domain name farklı gösterilir.

\$PWD → Çalışma diziminin tam yolunu gösterir.

Setting Environment Variables

- "varname = value" ile değişkene değer atılır.
- "PS1 = \$USER\$HOSTNAME" Orjinal shell prompt'u değiştirir.
- "PS1 = bash - Prompt"
- "PATH = \$PATH: \$HOME/bin"
- "PATH = \$PATH: ~: \$PATH"

• "DATE = 'date'"

Komutun ürettiği outputu DATE değeri olarak

* ('') kullanırsak arada ismi geçen komutu

('') veya ("") kullanırsak içindekini string olarak yazdın.

Textual Completion

<tab> Aktif komut ya da dosya adını tamamlamaya çalışır.

pu<tab> Alternatifleri verir.

push<tab> ile pushd <space>
geleştirilmiş.

Aliases

"alias shortcut = command" şeklinde her defasında uzun komut ismi yazmamak için kısa isim atarız.

Örnek alias l = "ls -F -C"

↓ ↘
kısa komutun
isim asıl ismi.

Command History

Gecmiş komutları listeler.

"fc -l <m> <n>" önceden yazılmış komutları m'den n'ye doğru listeler.

up ve down cursor tuşlarıyla history listesinde yukarı ve aşağı gidebiliriz.

Editing on the Command Line

bash, bazı satır düzenleme komutları sağlar.

Emacs-mode komutları:

- Esc -b → 1 kelime geri gel.
 - Esc -f → 1 kelime ileri git.
 - Ctrl -a → satır başına git.
 - Ctrl -e → satır sonuna git.
 - Ctrl -k → kursörden satır sonuna kadar texti yoket.
- esh* {
- set -o vi ile vi komutlarını kullanabiliriz.
 - set -o vi -tabcomplete TAB ile tamamlama sağlar.

Login Scripts

Bir sisteme her bağlandığımızda aynı yapılandırmaları tekrar yapmak için bu yapılandırmaların sisteme login olduğunda yapılmasını sağlar.

Login esnasında çalıştırılan scriptler:

- /etc/profile (mutlaka çalıştırılır)
- ~/.bash_profile (çalıştırılır)
- ~/.bash_login (↙ yoksa çalıştırılır)
- ~/.profile (↘ yoksa çalıştırılır)

Login'den sonra çalıştırılan scriptler:

- ~/.bashrc

Logout

- ~/.bash_logout

Example .bash_profile (partial)

- .bash_profile

umask 022 (0666 & ~022 = 0644 = rw-r--r--)

- .bashrc

If [-f ~/.bashrc]; then . ~/.bashrc. fi

- warm fuzzy environment file variable settings

→ export CVSROOT=~/.cvsrc → export PAGER
= /usr/bin/less
→ export EDITOR=/bin/vi

Example .bashrc (basically a saltwater)

- alias bye=logout
- alias l='ls -F -C'
- alias ps=psop
- alias h=history
- alias ll='ls -L -l -F'
- alias pw=pwd

Login Scripts

- csh /tm Login shell

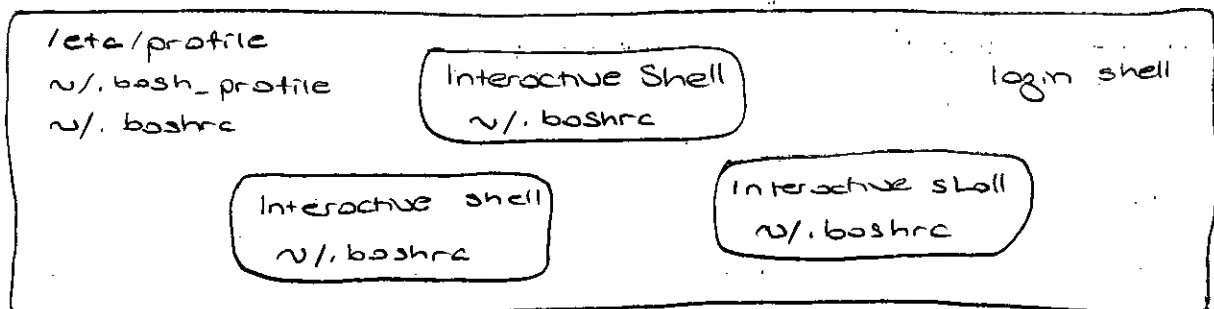
~/.profile

scriptm: /etc/cshrc

- ENV settings: by default her, gen terminal
tm: /etc/cshrc

→ ENV=\$HOME/.cshrc → EXPORT ENV (for bash)

Login and Other Shells



⑧ Basic Shell Scripting

Shell script komutlardan oluşan bir text dosyasıdır. Bu komut shellin bir komutymuş gibi işaretilir.

Bu komutlar

komut satırında yazıldığı gibi herhangi bir şeyi

shell değişkenlerini

kontrol ifadelerini (if, while, for...)

işerebilir.

Script Execution

"bash my-script" ile argumentmiş gibi veya

"#!/bin/bash" ile shellin ilk komut satırında koşulu.

Simple script

```
#!/bin/bash
```

```
echo "Hello, world!"
```

```
path = $ (pwd)
```

```
echo $path
```

sonuç:

```
Hello, world!  
/usr/home/smirt
```

Shell Variables

Değişkenler

- numeric değer,

- fonksiyonlar,

- string,

- read-only,

- dizin,

- komut satırı argümanları. -elabilir.

var → variable'yi ifade etsim.

\$var

var=100 → değeri 100'e setter

echo "\$+= \$+" → \$+=100 yazor

Numeric Variables

bash reel sayıları desteklemez. Numerik değer olarak sadece integer sayılar vardır.

"declare -i var=100" şeklinde deklarasyon ve değer atama yapılır.

C programlarındaki gibi numeric expressionler 2 parantez içinde gösterilir.

((exp)) → komutlar için

[exp] → değişkenler için

Ör ((var+=1)) or ((var++))

((var2=1+var)) or ((var2=1+\$var))

echo \$((var*7)) or echo \$[var*7]

★ C/C++'daki tüm operasyonlar uygulanabilir.

+, -, *, /, %, &, |, <, >, <=, >=, ==, !=, &&, ||

String Variables

" " arasındaki değerler string olarak düşünülür.

((var2=1+\$var))

ise integer olduğunu gösterir.

Bir string üzerinde işlem yapmak için şu syntax kullanılır.

$\$(string:n)$

Bu komutla stringin ilk n karakterinin otılmasını söyler.

$\$(string:(-n))$

Bu komutla stringin son n karakterini söyler.

$\$(string:n:m)$

Bu komutla $(n, \underbrace{1 \text{ ile } (m+2)}_{\text{harc}})$ arasında bulunan karakterleri verir.

$\$(\#string)$

Bu komutla stringin uzunluğunu alırız.

$string-ua-1=\$string-ua-1 \ \$string-ua-2$

Bu komutla stringin değerini alıyor.

sağıdaki 2 string arasında boşluk kalır.

Variable Expansion

$\${name}$

→ Basit

$\${name}:-word$

→ name setlenmemişse wordu kullan.

$\${name}:=word$

→ name setlenmemişse wordu kullan ve ^{ayar}

$\${name}:[word]$

→ name setlenmemişse 'word' errorunu

$\${name}:+word$

→ name setlenmişse wordu setlenmemişse null kullan.

$\${name}%word$

→ en küçük son ek patternini çıkart.

$\${name}%%word$

→ en büyük son ek patternini çıkart.

$\$ \{name\#word\}$ → en küçük örnek poternini çıkart.

$\$ \{name##word\}$ → en büyük örnek poternini çıkart.

Array Variables

Array bir değerler listesidir. size atanmak zorunda değildir.

• $\$ \{name[index]\}$ → bir değeri referanslar.

$\$ \{a[3]\}$ → 4. pozisyonundaki değer

$\$ a$ → 0. pozisyonundaki değer $\equiv \$a[0]$

• declare -a sports → diziye ifade eder.

sports = (basketball football soccer)

sports[3] = hockey

• sports = { (football basketball) }

more sports = { \$ sports tennis }

• $\$ \{array[@]\}$ } Dizinin tüm karakterlerine erişir.
 $\$ \{array[*]\}$

• $\$ \{\#array[*]\}$ → Dizinin boyutunu verir

• Dizilere komutların çıktılarını da atayabiliriz.

$\$ files = [ls]$

echo $\$ \{files[*]\}$

Exporting Variables

Child süreçler oluşturulduğu zaman parent içerisindeki değerlere child süreçler içinde de erişilebilmesini sağlar. ("export var" ekleyerek)

Command Line Argument

Argumentler scripte ulaşıyorsa herbirine \$1, \$2, \$3, etc. değerleri verilir.

\$0 → scriptin adı

\$* → \$0 hariç birbirinden boşlukla ayrılmış argümanların stringi

\$@ → \$0 hariç argümanların dizesi

\$# → argüman sayısı

Output and Quoting

Standart output mesajı göndermek için

echo a message

Kullanılır:

\$ → Değişkenlerin değerleri

' → Komutların kosumu gösterilebilir.

-n → Bir prompt dur.

Ör: echo -n "Yes/No?"

Ör: echo " !date +%D" # 04/30/05
date'nin hangi formatta
gösterileceği.

\ escape karakterleri (\$, ", ...) kullanmak için.

Return Values

Scriptler integer değer geri dönebilirler. Bu geri döndürülen değeri \$? ile erişebiliriz. Bu değeri geri döndürmek için return N kullanabiliriz. Bu daha çok koşullu ifadelerde kullanılır.

En son durum hakkında bilgi edinmek için yayı

0 → komutun başarıyla çalıştığını gösterir

↓
return
value

Bu değer aslında bizim çalıştığımız sistemin koda yazıldığı içerisinde bulunan exit komutunun parametresine konulmuş gelir.

Shell Variables

3 sınıfa ayırırız:

Kullanıcı tanımlı değişkenler: ör: age

Environment değişkenler: PATH, HOME, TERM, PS1, ...

Predefined değişkenler: Formal parametre gibi değişkenler önceden tanımlanmıştır ve bu gruba girer.

* Değişken isimleri alt tırtıklı yazılabilir ve alt tırtıklı ile başlayabilir.

ör: FRUIT, TRUST_NO_1, _2_TIMES

* arg = "15 -a -1"

Burada boşluk bracketmanalya.

Reading User Input

Standart inputu okumak için read kullanılır.

ör: \$ cat read_name

echo "Please enter your name and surname:"

read name1 name2

echo "Welcome to CE Dept, KTU, \$name1 \$name2"

Command and Arithmetic Expansion

stdout ile komutu yer değiştirmek için

var='command'

var = \$(command)

\$(expression) → expression değerini yer değiştirmek için

Computation on Variables

expr command

→ expr val1 op val2

→ expr \$val1 op \$val2

\$>expr 5+7 → 12

\$>expr 6-3 → 3

\$>expr 3*4 → 12
çarpma işi

\$>expr 24/3 → 8

\$>sum='expr 5+6' - \$>echo \$sum → 11

\$>echo 3*4
↓
çarpma

Görsel dizimindeki dosyanın adı

\$>a=12 \$>b=90 \$>echo sum is \$a+\$b → 12+90

Handling Shell Variables

\$1-\$9 → yerel parametreler

\$0 → komut adı

\$# → yerel argümanların sayısı

\$? → Son geçen komutun çıkış (exit) değeri (0,1,2...)

\$\$ → Shellin process ID'si

\$! → En son geçen komutun process ID'si

\$- → Shell açılırken shell'e verilmis optionlar

\$* → Shell'in tüm argümanlarını içeren string

\$@ → \$* ile aynı.

Notes!

- \$*, \$@ '' kullanılsa da kullanılmassa da de-

ğişmez.

- "\$*"

2

- "\$@"

Passing Arguments

Shell scriptlerine komut satırı üzerinden pa-
rametreler vermek için \$1 ve \$9 arasındaki
parametreleri kullanmaktayız. Komut ismi hariç
9 parametre olabilir.

Eğer 9'dan fazla parametre girilmeye o
zaman shift komutunu kullanabiliriz. Shift
komutu argümanları birer birer sola doğru
kaydırır. Bir kez shift kullanıldığında \$9'un
tutacağı değer \$10'un değeri olur, \$0 atılmış
olur.

Öz \$ cat pass-arg

```

ör & cat sum_orç
sum=0
while ( $# -gt 0)
do
    sum= 'expr $sum + $1'
    shift
done
echo sum is $sum

```

Conditional Exec. Operations

&& → (ör: command1 && command2) 1. komut başarıyla sonloldysa 2. komut اجرا edilir.

Bir komutun kabulbilmesi önceki komuta bağlı sa kullanılır.

|| → 1. komut başarısız oldysa 2. komut اجرا edilir.

Başarı 0, başarısızlık 1 ile gösterilir.

ör ls | grep "mydoc.doc" && rm mydoc.doc

dosya mevcutsa silme diziminden çıkar.

ör cat mydoc.doc || echo "file not found"

Dosya mevcut değilse bulunmadı mesajı ver.

Conditional Statements

Koşullu ifadeler uc bir kullanım. \$? ile gerçekleştirilir.

Conditions

((condition)) → integer kullanım

[(condition)] → string kullanım

ex) ((a==10))

((b>=3))

[[\$1 = -n]]

[[-e \$file]] → Dosya exist mi?

[[-f \$file]] → Dosya regular mi?

[[-d \$file]] → Dosya directory mi?

[[-L \$file]] → Dosya sembolik link mi?

[[-r \$file]] → Dosya read iznini sahip mi?

[[-w \$file]] → Dosya write iznini sahip mi?

[[-x \$file]] → Dosya çalışabilir mi?

[[-p \$file]] → Dosya pipe mi?

The if Statements

if test

then

commands (if conditions is true)

else

commands (if conditions is false)

fi

* else if yok elif var (elif test then command)

test Commands

Komutların başarılı ya da başarısız sonuçlandırıldığını gösterir.

- eq → equal
- ne → not equal
- gt → greater than
- lt → less than
- ge → greater and equal
- le → less and equal

Stringler için:

- = → equal
- != → not equal
- z → uzunluğun 0 olup olmadığı
- n → stringin en az 1 karaktere sahip olduğunun kontrolü

Dosyalar için:

- f → Dosya mevcut
- s → Dosya var ve boyutu 0 değil
- d → Directory mevcut
- r → Dosya var ve okunabilir
- w → Dosya var ve yazılabilir
- x → Dosya var ve çalıştırılabilir

For loops

for var [in list]

do

.. Commands

done

While loops

while command - list+1

do

.. command - list+2

done

* break, continue, return de kullanilabilir.

Until loops

until command - list+1

do

.. command - list+2

done

* do ---- done

if ---- fi

case ---- esac

} ... yerlere konut yazilir.

Select loop

select name in word1 ---- wordN

do

list

done

selectin kullandığı özel kere değişkenleri vardır. \$REPLY kullanıcının seçimini verir.

using basename

Absolute veya relative path üzerinde dosyanın ismini ortaya çıkarır.

ÖY \$> basename /usr/bin/sh

sh

using printf

echoya benzer. Ama echo 'n' karakterini ekler, printf ekmez.

Formatlanmalıdır.

printf format arguments

% [-]m.nx

x → formatlama dizisinin tipi

m → min dol uzunluğu

n → max dol uzunluğu (kullanılmaz olarak kullanılır)

(-) → Alın işerisinde kullanılmada kullanılır. (default) ^{sağa} _{yanıya}

• x: stringler için s

karakterler için c

integerlar için d

hexal sayılar için x

octal sayılar için o

özel floating pointler için e

sabit floating pointler için f

şiki floating pointler için g

olarak
kullanılabilir.

örü#!/bin/bash

```
printf "%32s %s\n" "Filename" "File Type"
```

```
for i in *;
```

```
do
```

```
printf "%32s" "$i"
```

```
if [-d "$i"]; then echo "directory"
```

```
elif [-h "$i"]; then echo "symbolic link"
```

```
elif [-f "$i"]; then echo "file"
```

```
else echo "unknown"
```

```
fi;
```

```
done
```

* yerne gollama drahimdeki dosya isimleri gelir
tee command

pipelara yazilan o degerleri bir dosyaya yazmak ism kullanilir.

```
$> date | tee now
```

```
$> ls -l | tee list | wc
```

```
$> ps -oel | tee processing | grep
```

```
"$UID"
```

18 satiri 18 wordu 134 karakteri oldugunu verir.

Formatted Output

Directories: RCS, dev, humor, images, install, java

File: index.html.

Associating Files with a File Descriptor

Unix sistemlerinde her bir kovan süreç 3 standart file descriptor default olarak verilir.

Standart Input (STDIN), 0

Standart Output (STDOUT), 1

Standart Error (STDERR), 2

Biz default olarak bu dosyaları belirleyicilerin terminal ile ilişkilendirildiğini biliyoruz.

Bir dosyayı yazmak ismi açmak istediğimizde exec komutunu kullanabiliriz.

exec n > file → dosyaya yaz. (bir satırdan itibaren)

exec n >> file → dosya sonuna ekle

Ör \$exec 4 > fd4.out

Bu komutun sonro 4 fd4.out dosyasını belirtmek ismi kullanılır.

General Input/Output

command 1 > file.	} output
command 1 >> file	
command 0 < file	} input

Redirection to separate files

Ör command 1 > file 1 2 > file 2

Bu işlemle STDOUT file 1'e, STDERR file 2'ye yönlendirilmiştir.

Return Value

```
function add_two {  
    ((sum = $1 + $2))  
    return $sum  
}
```

add_two 1 3

echo \$?

Fonksiyonun sonucu ile toplanarak toplanan değeri belirten değeri geri döndürür.

echo Command

echo [options] [string, variables...]

- n Yazdıktan sonra satır atlama.
- e Postittekileri kullanmayı sağlar.

Display colored text

\$> echo "\033[34m Hello!"

Hello! → mavi

\033 → bazı aksiyonlar olan escape karakteri

34 → parametre (mavi renk)

[→ CSI'yi başlatır,

m → actionu belirler.

Genel syntax: echo -e "\033[escape-code message"

h ANSI modu setler.

g numlock, capslock...

l ANSI modu resetler.

s cursorun o eski pozisyonunu saklar.

m Renkli ve bold gösterir.

u cursor pozisyonunu eski haline getirir.

10 Software Development: g++ and make

49

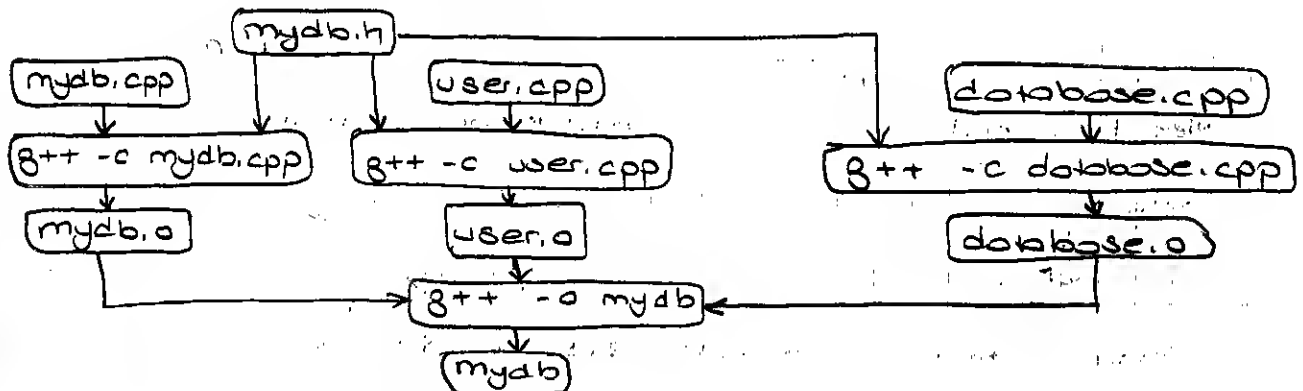
Seperate Compilation Scenario

source files → mydb.cpp, user.cpp, database.cpp

common headerfile → mydb.h

executable → mydb

Build Dependencies



using make in compile

Eğer bir kaynak kodun derlenmesi birden çok dosyanın işerliğini değiştiriyorsa make kullanılır.

Basic Operation of make

[mm] makefile

Eğer derlenecek program başka bir kaynak koda bağımlıysa öncelikle o kaynak kod derlenir.

mydb: mydb.o user.o database.o

⇒ g++ -o mydb mydb.o user.o database.o

mydb.o: mydb.cpp mydb.h

⇒ g++ -c mydb.cpp

user.o: user.cpp mydb.h

⇒ g++ -c user.cpp

database.o; database.cpp mydb.o
target

⇒ g++ -c database.cpp

Port of a makefile

Dependence lines;

- Hedef isimleri ve bağımlılıkları içerir.
- Başya ve hedef bağımlılıkları.

Commands;

- Daima tab. karakteriyle başlar, Dependency line'nin altındadır.

macros and special Variables

MACRONAME = macro_value → tanımlama

\$ {MACRONAME} → kullanım.

\$@ → hedefi temsil eder. \$? → bağımlılıkları temsil eder.

Invoking make

make → ilk targeti inşa eder

make targets → tüm targetleri inşa eder.

-n komutları koyma, -f<file> <file>'i [mm]akefile yerine kullan

Suffix Rules

\$ {CC} \$ {CFLAGS} -c \$< → file.cpp'yi temsil eder.

* Komutlar # ile başlar.

* Komut satıra sığmazsa \ ile alt satıra geçilebilir.

Other Useful makefile Tips

- clean: `rm -f mydb`
`rm -f *.o`

exe ve obj kodlarını silen komutlar:

(11) Debugging

Why use a debugger?

Hata tespitinde kullanılır. (Programın satır satır kodları ya da breakpoints kullanılır).

Debugging with gdb

-g kullanılmaktadır.

Basic gdb commands

Help	help
Breakpoints	break main
Running	run arg-list
Step to next line	next
Step into functions	step
Continue running	cont
List source	list
Quitting	quit

- Running GDB with a core dump
- NetBSD: `gdb name name.core`
- Solaris: `gdb name core`

Execution commands

list or l → list

→ list main

→ list 56

run or r → run

→ run file.txt file2.txt

next or n → bir sonraki satıra koama

step or s → bir sonraki satıra geçme

Breakpoint commands

break or b → bir breakpoint setle

break 10

break main

delete or d → bir breakpoint silme

delete

delete 2

continue or c → durdurulmuş koamayı devam ettir

Program information commands

print or p → değeri yazar

print x*y

print function(x)

display → görüntülenen değere devam et.

undisplay → görüntülenen değeri sil.

where → aktif fonksiyonun stackini göster.

Miscellaneous Commands

set → dəyəri dəyişdir.

set n=3

help or h → help mənasını göstər

help

help step

help breakpoints

quit or q → qolb'dən çıx.

~~I. VİZE~~
SONU

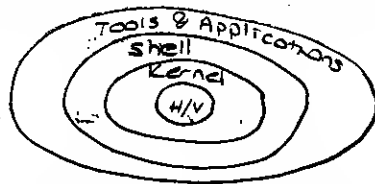
Sinav qəzəmləri

1. s> * → "13" yazor.
- q> '*' → "aa b cc" yazor.
- q> ? → b scripti kəzor. "123" yazor.
- q> '?' → "bosh: 123; command not found" yazor.

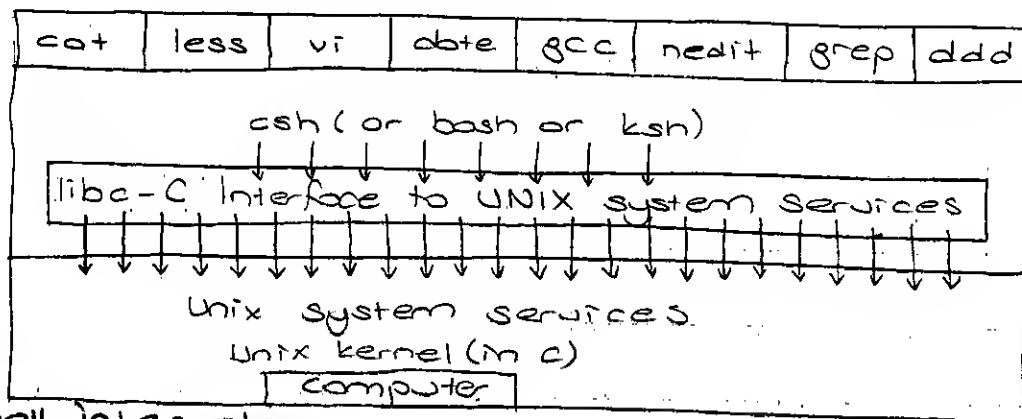
2.

2

13. File Management



Parts of UNIX OS



Shell Interpreter

Shell C'de yazılmış bir programdır. Komut satırına yazdığımız

```
#!/bin/sh
```

ile shell yorumlayıcısını kışoriz.

Parser treadi syntax analizi yapar, komut satırındaki veriyi anlamlı parçalara ayırır ve düzenimini kontrol eder, Parse işleminden sonra programın nasıl işra edileceğini belirler.

Standard Libraries

Standard kütüphaneler içinde sistem sağrılar, bu lunmaz. Bunlar doğrudan kernel tarafından yürütülen ve OS'ye erişim sağlayan program parçalarıdır.

library modülleri genellikle birçok obje dosyasından (makine dilindeki kodlamayon, etiket parçalarına sahip dosyalar) oluşur. Bir library dosyası örneklem:

or crv mylib. a *.o

yazılır.

Sistem libraryleri /usr/lib.ve /usr/local/lib'de bulunur.

Standard Libraries

.a libraryleri paylaşılmaz, kullanılacak exe kod içine yazılır. Eğer sistemde aynı libraryi kullanacak birçok kod oluşturulacaksa bu libraryler birçok kez tekrarlandığından ana bellekte zıame olur. Eğer yandan performans artımı sağlar.

.so libraryleri paylaşımlıdır. Ana bellekte sadece 1 kopyası vardır. Tam süreçler bu kopyayı kullanır. Bellek kullanımını azaltır ve performans düşüşüne sebep olur.

System Calls

libs modülü ile dolaylı yoldan sistem çağrılarını yaparız. libs, sistem çağrılarında C arayüz sağlar. C kaynak kodu içerisinde libraryleri kontrol eden sistem çağrılarını öğrenabiliriz.

System callar 4 ana gruba ayrılır.

- Dosya yönetimi
- Sistem yönetimi
- Hata ve sinyal yakalama
- Haberleşme

Sistem çağrılarını bize truss gösterir.

System Call Routine

— ASSEMBLY ve EXPLANATION SİYATTAN GALS. —

Executing a Program

crto özel bir start-up routine'idir. Her zaman programa link edilir. Programa geçen argümanları alır ve maini çağırır.

libc kütüphanesi otomatik olarak kurulmuş kütüphane (c) link edilir. Böylece printf, open gibi komutlar tanımlanıyor.

C versus C++

C'de stringler karakter dizileri olarak görülür. Deklarasyonlar her komutun başında olmak zorundadır, mikas alınmaz.

C++'da

printf yerine cout

scanf ve fgets yerine cin

kullanılır.

Basic File I/O

Unix'te herşey (printer, terminal, ...) bir dosya olarak tutulur. Bu her bir devicenin kullanımı, okuma yazma dosyası okuma yazma gibidir. Dosyalar okuma ve yazma işlemi yapılabilir.

I/O sistem çağrılarını kullanmak için `<stdio.h>` dosyasını sistem çağrılarını eklenmelidir.

Birçok UNIX I/O

- open
- close
- read
- lseek
- write

ile başarılabılır. Her bir sistem çağrısı bir file descriptor (integer) temsil edilir.

3 dosya otomatik olarak açılır.

FD0 : standart input

FD1 : standart output

FD2 : standart error

Bu 0,1,2 değerleri init sırasında gelir. Bu

3 açık dosyayı shell miras alır ve ihtiyaç olan süreçlere verir.

Bir dosya açıldığında en küçük serbest FD'yi alır.

man -s 2 <systemcall> (daha fazla bilgi için).

open()

`fd = open(path, flags, mode);`

path: char*, absolute or relative path

flags:

O_RDONLY → okumak için aç.

O_WRONLY → yazmak için aç.

O_RDWR → okumak ve yazmak için aç.

O_CREAT → bu dosya yoksa oluştur.

O_TRUNC → dosyanın içeriğini sil.

O_APPEND → dosya sonuna veri ekle.

mode: İzinler arasında O_CREAT bulunmalı.

Geniş dâhâretten değış dosya tanımlayıcısı ismini alır:

`fd = open("myfile", O_CREAT, 00644);`

Öğ Bu dosya oluşturulmuştur. Okuma ve yazma yapılabilirken aynı bu izinler de eklenmelidir.

read()

Dosya açılıp bir FD elde edildikten sonra bu FD'yi diğer sistem çağrılarında (örneğin read, write...) kullanılır.

`bytes = read(fd, buffer, count);`

fd: FD'nin adı

buffer: count byte arraya pointer

count: Okunabilecek max byte sayısı.

Bu satır okunan byte sayısını veya -1 (error) döndürür.

```
Ör int fd = open("somefile", O_RDONLY);
```

```
char buffer[4];
```

```
int bytes = read(fd, buffer, 4);
```

```
write()
```

```
bytes = write(fd, buffer, count);
```

Buffer içeriği dosyaya yazılır. Geri döndürülen değer yazılan karakter sayısı ya da -1 (error)'dır.

fd, buffer ve count'un anlamları read()'daki ile aynıdır.

```
int fd = open("somefile", O_WRONLY);
```

```
char buffer[4];
```

```
int bytes = write(fd, buffer, 4);
```

```
close()
```

```
return_val = close(fd);
```

Bu satırda fd dosya tanımlayıcısı serbest bırakılır. Başarılı ise 0, başarısız ise -1 değerini geri döndürür.

* Sadece exit() sistem çağrısı geriye bir değer döndürmez.

lseek()

retval = lseek (fd, offset, whence);

file pointer'in yerini değiştirir.

offset: byte sayısı

whence:

SEEK_SET → Dosya başlangıcından offset kadar ileri

SEEK_CUR → Mevcut pozisyonun offset kadar ileri

SEEK_END → EOF'dan offset kadar ileri

Dosya başlangıcından offseti veya -1 (error) geri döndürür.

EKSİK >

File I/O using FILES

Unix programı yüksek seviyeli G/I/O fonksiyonları içerir,

fopen()

fread()

fwrite()

fclose()

fseek()

Bunları kullanmak için include <stdio.h> gerekir.
Bunlar FD değil FILE datatipi kullanılıyor.

copying std in to stdout

define _BUFSIZE 8192

int main(void) {

int n;

char buf[_BUFSIZE];

while ((n = read(STDIN_FILENO, buf, _BUFSIZE)) > 0)

if (write(STDOUT_FILENO, buf, n) != n)

printf("write error");

if (n < 0)

printf("read error");

}

I/O Efficiency

BUFSIZE artıkkı

kullanıcının CPU kullanımı azalır

sistemin CPU kullanımı azalır

clock zamanı azalır

... - 2 azalır.

Executing a Program

UNIX sistemlerde programlar exe sistem çağırışı ile اجرا edilirler. Bunların birçoğu kernel var. Mevcut adres alanı üzerinde yeni bir program koşulur.

exec başarılıysa geri dönmeyiz, başarısızsa -1 döndürür.

Örnek Program X: int main () {

int i=5;

printf ("%d\n",

exec

printf ("%d\n",

}

Program Y: printf ("hello\n");

exec () properties

exec ile yeni program çalıştığı zaman bu an-
daki süreçten miras alınır. Miras alınan veriler adres
alanındaki veriler değildir, sürecin adresini alarak
ran verilerdir. Bu veriler:

- Sürecin PID'leri, UID'leri, GID'leri ve session ID'leri.
- Sürecin açtığı dosyalar ve FD ler.
- Kontrol eden terminaller
- Herhangi bir işlem için sinyal ve alarmlar.

- cwd, root dizini, kaynak limiti,

- dosya kilitleri,

- dosya modu oluşturma maskesi,

exec() versiyonları

~~exec~~(char *path, char *arg0, ... (char *)0);

execv(char *path, char *argv(1));

execl(char *path, char *arg0, ... (char *)0, char *envp(1));

execve(char *pathname, char *argv(1), char *envp(1));

execvp(char *file, char *arg0, ... char *);

execvp(char *file, char *arg(1));

Waiting for a Process to Terminate

Bir child adres oluşturulduğunda child ve parent zaman paylaşımı ya da eş zamanlı olarak çalışır. Parent childin terasını tamamlamasını bekler.

int wait (int *status)

waiti çağırarak adres tüm childlar koştuktan sonra bloklanır, hiç child processle sahip değilse bekler. geri döndürür, childin koşturması sonlandıysa, childin termination durumunu geri döndürür.

Signals

Yazılımla gerçekleştirilen bildirimlere verilen isimdir. Bunlar

- floating point hataları
- childin ölmüşü
- alarm clock çalındığında
- control-c
- control-z

Bu sinyaller sistem tarafından da herhangi bir process tarafından da gönderilebilir.

What are signals for?

Sinyaller süreçler arası senkronizasyonun bir biçimini sağlamak için kullanılırlar.

İcra sonlanması parent süreçte bir child sinyali ile belirtilir.

Processler arası veri iletimi için pipeline ve socketler kullanılır.

Sinyaller makine kesmeleri ve

x

2

Software Interrupts

Sinyaller bu adı da verebiliriz. Bunun procedurleri `<sys/signal.h>` şeklinde listelenmiştir. kill-1 tüm işletim sistemleri tanımlanmıştır.

2

Signal Table

Burada sinyaller tam yapılacak işlemler bulunur.

örneğin: \times

2

Sending a signal

Komut satırında kullanım:

`kill [-signal] pid [pid]...`

Bu sinyal olan süreçler genellikle donatıcılar

Signalling between processes

`kill (int pid, int sig);`

2

Kill Usage

`retval = kill (pid, signal)`

→ `pid > 0` ⇒ Bu process

→ `pid = 0` ⇒ Bu gruptaki processlere

→ `pid = -1` ⇒ Tüm processlere

} gönderilir

• `root` ⇒ Tüm sistem processlere

• `!root` ⇒ Aynı uiddeki processlere

What happens at "signal time"?

kill sinyolını alan süres içerisinde sonlandırmak,
stop sinyolını alan ise durdurmak sırasında, Sm-
yolu reddedilince.

2

Signal Handling

```
void (*signal)(int, sig, void (*func)(int), )()
```

Sistem geçişiyse ilgilenmek için bu fonksiyon
kullanılır. Burada func SIG_DEL veya SIG_IGN ola-
bilir.

Timer Signals

SGLALRM

SGUTALRM

SGP

2

Advanced Signal Interface

Symbolle - ilgilanmek tam daha gelismis bir a

raysedu.

#include